Aufgabe 1

Ändern Sie die Syntax von WHILE, indem Sie INTEGER- und REALZahlen unterscheiden.
In der Vorlesung wurden die ganzen Zahlen wie folgt definiert:

```
// ganze Zahlen (endlicher Ausschnitt der ganzen Zahlen MIN+1 .. MAX)
Z::= 0 | 1 |...| MAX | -1 | -2 |...| MIN
```

Gleitkommazahlen können in WHILE so abgebildet werden:

```
// natürliche Zahlen mit 0
N::= 0 | 1 | ... | MAX
// reelle Zahlen
R::= Z.N
```

Die Definition der reellen Zahlen muss zur Vollständigkeit in die Konstanten und Terme eingetragen werden.

```
K::= Z | R | W
T::= Z | R | I | T1 OP T2 | read, für T1, T2 in TERM
```

von Hinnerk: Imho sind das so nur die rationalen Zahlen, da es ja reelle Zahlen gibt, die nicht als Bruch darstellbar sind (z.b $\sqrt{2}$). Daher würde ich etwas wie das folgende vorschlagen:
von Tobi: Ja, wesentlicher Punkt! ;) Ich habe meine irrationalen Zahlen auskommentiert. Bei deiner Lösung bist aus meiner Sicht etwas durcheinander gekommen. So wie ich das sehe könnte man bei dir auch sagen GA = Z. Bei deiner Lösung ist es möglich o.-1 zu erzeugen. Hinter dem Komma dürfen nur natürliche Zahlen stehen.

(i) **Elementare Einheiten**

$ZIF ::= 0\|1\|...\|9\|0Z\|1ZIF\|...\|9ZIF$	Ziffern
$GA ::= ZIF\| - ZIF$	Ganzzahliger Anteil
$R ::= GA.Z$	REAL
$Z ::= 0\|1\|\cdots\|Max\| - 1\| - 2\|\cdots\|Min$	ZAHL= $Min, ..., Max$
$W ::= \underline{true}\|\underline{false}$	BOOL= $wahr, falsch$
$K ::= Z\|W$	KON=ZAHL∪BOOL
$I ::= a\|b\|\cdots\|z\|a_1\|\cdots\|z_1\|\cdots\|\cdots$	Id=Bezeichner
$OP ::= +\| - \| * \|\%\|\underline{mod}$	OP (arithmetische OP)
$BOP ::=< \| \leq \| \geq \| = \| \neq \| >$	BOP (Vergleichsops)

(ii) **Zusammengesetzte Einheiten**

$T ::= Z\|I\|R\|T_1\ \underline{OP}\ T_2\|\underline{read}$, für $T_1, T_2 \in TERM$	TERM
$B ::= W\|\underline{not}\ B\|T_1\ \underline{BOP}\ T_2\|\underline{read}$, für $T_1, T_2 \in BT$	BT boolscher Ausdruck
$C ::= \underline{skip}\|I := T\|C_1; C_2\|\underline{if}\ B\ \underline{then}\ C_1\ \underline{else}\ C_2\|\underline{while}\ B\ \underline{do}\ C\|\underline{output}\ T\|\underline{output}\ B$	COM

PS: Die Verwendung des Formats "meines" Skripts ist nicht als Affront gedacht, sondern resultiert daraus, dass ich zu faul war, es in Tobis Format zu transferieren. (Tobi: wayne)

Aufgabe 2

Definieren Sie für eine geeignete Erweiterung der Sprache WHILE eine konkrete Syntax, die eindeutig ist.
Zusätzlich zum If-Else wäre Switch:

```
C::= switch T: CASE
CASE::= CASE CASE | case B do C | default C
```

Anmerkung Hinnerk: Sehr gute Idee, aber ich glaube das reicht formal so noch nicht. Daher folgender Vorschlag:

```
CASE::= case B -> do C | CASE:CASE | skip
C::=skip|I:=T|C_1;C_2| if B then C_1 else C_2| while B do C| output T|
    output B| switch T: CASE
```

Formal muss alles aufgeführt sein (nitpicking, ich weiß), aber in deiner Ursprünglichen form ist es im Prinzip nur ein if (sogar ohne else). Man muss es so bauen, dass man es durch einsetzen zu jedem gewünschten Ausdruck expandieren kann. Bei mir kann es so wie bei dir sein (-> statt : hab ich nur gemacht, damit es deutlich ist, dass da ein Unterschied ist). CASE kann so aber entweder ein case sein, oder beliebig viele (durch CASE:CASE) und das skip erspart ggf. notwendige Fehlerbehandlung, wenn kein case matcht (denke ich jedenfalls).
von Tobi: Verstehe, bei genauerem überlegen würde ich aus dem skip ein default C machen.

Eine weitere wirklich sinnvolle Erweiterung wäre das Programm. Es besteht aus einem oder mehren Befehlen. Weil die Menge der Befehle bereits ausreichend definiert ist sieht diese Erweiterung ziemlich unspektakulär aus:

```
//Programm
P::= C
```

Anmerkung Hinnerk: trivial, aber iirc war das im letzten Jahr eine korrekte möglichkeit

Aufgabe 3

Formulieren Sie informell eine Präzisierung der angegebenen WHILE-Semantik, die die genannten Fehlerquellen:
Anmerkung Hinnerk: Ich denke es soll die Fehlerbehandlung definiert werden und nicht beschrieben werden, wie sich der Fehler äußert
Anmerkung Tobi: Ja, dachte ich auch, nur dazu muss man die Fehlerquelle (informell) spezifizieren. (So behält man sich die Chance vor, das man seine eigene informelle Spezifikation erfüllt hat)

Bereichsüberschreitungen 1. Man verlässt z.B. den definierten Bereich der ganzen Zahlen, also MIN-1 oder MAX+1:
Hinnerk: Nachfolger von MAX ist MIN, Vorgänger von MIN ist MAX. Ergebnis ist, dass man damit das aus z.B. C gewohnte Integer-Overflow-Verhalten bekommt.
2. Über das Ende einer Zeile (einer Datei) hinaus lesen:

Division durch Null Wenn T1 / T2 ein Ausdruck ist, darf T2 keine 0 sein.
Hinnerk: Da Division durch Null nicht definiert ist, wird ein Fehler geworfen und die Programmausführung abgebrochen.

Berechnung von read bei leerer Eingabedatei Wenn read eine leere Eingabedatei erhält, dann wird read entweder als skip oder Identitätsfunktion ausgeführt.
Hinnerk Anmerkung: So hat man aber immernoch uninitialisierte Werte in den Variablen, was imho zu undefiniertem Verhalten führt. Daher schlage ich folgendes vor:
Hinnerk: Im Falle einer leeren Eingabedatei liest `read` für B `false` und für Z `0`.

Typkonflikte Eine Zahl mit dem Rückgabewert von read addieren, wenn in der Datei ein Wahrheitswert drin steht. Es muss zunächst geprüft werden, ob der Typ von T1 mit dem Typ von T2 kompatibel ist. Darüber hinaus muss die Operation auf T1 und T2 definiert sein.
Hinnerk: Im Falle von `read` gibt es keine Typfehler, da Variablen in WHILE nicht getypt sind. Ansonsten sehe ich zwei Möglichkeiten: wird ein B erwartet, aber ein T oder ein Z gelesen, wird 0 als $false$ und alles andere als $true$ verstanden (analog zur C Semantik). Alternativ führt jeder Typfehler zu einem Fehler und das Programm wird an der Stelle des Fehlers abgebrochen.

behandelt.